EOLSR : Energy efficient OLSR, un protocole de routage économe en énergie pour la présérie

Cédric Adjih, Pascale Minet, Ichrak Amdouni, Ridha Soua

Version 2

15 Décembre 2011

INRIA
Rocquencourt,
78153 Le Chesnay Cedex

Table des matières

Ce document spécifie le routage EOLSR tel qu'il sera implémenté dans la présérie. Ce document vient mettre à jour les documents :

- "EOLSR : Energy efficient OLSR, un protocole de routage économe en énergie pour la présérie" du 3 Mars 2011.
- "EOLSR : Energy efficient OLSR, un protocole de routage économe en énergie : mise-à-jour des spécifications du mode général" du 3 novembre 2009.
- "Spécifications d'EOLSR en mode stratégique : un protocole de routage économe en énergie pour applications avec noeuds stratégiques" du 15 juin 2009.

Les précisions et modifications portent sur les points suivants :

- la gestion des voisins est simplifiée. Pour des raisons de passage à l'échelle dans un réseau de capteurs, les voisins à 2 sauts ne sont plus maintenus. Il s'en suit qu'il n'y a plus de relais multipoint (ni MPR, ni EMPR) ;
- le calcul du coût pour la sélection des routes économes en énergie est lui aussi simplifié ;
- le format des messages est allégé ;
- la stabilité des arbres devant être coloriés est contrôlée.

1 Présentation d'EOLSR

1.1 Routage selon MaCARI ou selon EOLSR

Dans OCARI, il existe deux techniques de routage :

1. selon EOLSR, le protocole de routage, le message suit la route de moindre coût énergétique et tient en compte de l'énergie résiduelle des noeuds ;
2. selon MaCARI, le protocole d'accès au médium, le message est acheminé à son destinataire selon la structure de l'arbre établi par MaCARI.

Le choix entre les deux techniques s'effectue selon la règle suivante :

- Lors de sa demande d'envoi d'un message vers une destination, une source doit spécifier s'il s'agit d'un transfert contraint en temps.
  - Si tel est le cas, le routage selon l'arbre établi par MaCARI est utilisé,
  - Sinon, le routage selon EOLSR est utilisé.
- Cette information sera spécifiée au niveau deux, dans l'entête du message, afin d'être connue par chaque noeud qui recevra ce message.

1.2 EOLSR dans la présérie

EOLSR est l'extension du protocole de routage OLSR économe en énergie. Dans ce document, nous présentons une version de EOLSR adaptée aux applications de collecte de données, appelées aussi stratégiques. EOLSR est responsable de la construction d'un arbre de routage dont la racine est un noeud stratégique, c'est à dire un puits de collecte de données.

EOLSR conserve certains principes d'OLSR mais apporte des simplifications au niveau des tables maintenues, des traitements effectués (simplification du calcul des coûts des routes, suppression des relais multipoint) et du format des messages échangés. Ainsi :

- Pour permettre un meilleur passage à l'échelle, un noeud ne maintient que la table de ses voisins à 1 saut. Il ne conserve pas de table pour ses voisins à 2 sauts. Il s'en suit que le concept de relais multipoint n'est plus utilisé.
- La sélection des routes est modifiée afin de retenir la route qui évite les noeuds de faible énergie résiduelle et minimise le coût énergétique dénoté Cost(path). Ce coût est simplifié afin de rendre son calcul plus facile dans l'implémentation.
- Le format des messages est simplifié. Il n'est plus envisagé de grouper plusieurs messages en un seul paquet. Ceci simplifie l'entête.
- Un nouveau message est introduit pour informer la racine de l'état de la topologie de l'arbre, en particulier contrôler la stabilité de tout arbre devant être colorié.
  Cette fonctionnalité représente un premier niveau d'interaction entre EOLSR et SERENA, l'algorithme d'ordonnancement de l'activité des routeurs par coloriage. L'échange de ce message permet à la racine de l'arbre de s'assurer de la stabiltié des liens dans cet arbre, pour pouvoir lancer le coloriage. De plus, la racine sera informée via ce message qui est remonté sur l'arbre de routage des changements de la topologie. Cela lui permettra par exemple, de relancer le coloriage sous certaines conditions, si celui-ci devient invalide.
  Par conséquent, dans ce document, nous distinguons **deux types d'arbres dont la racine est un noeud stratégique** :

  1. les arbres devant être coloriés. Un arbre de ce type doit être stable avant d'être colorié.
  2. les arbres ne devant pas être coloriés.

Par ailleurs, nous introduisons une option : **l'option backup**.

- Dans le cas général, tout noeud maintient un parent (prochain saut) dans l'arbre dont la racine est un noeud stratégique.
- Si l'option backup est utilisée, tout noeud doit en plus maintenir un backup de ce parent. Ceci permet d'accélérer la réparation de la route lorsque le lien avec le parent ne peut plus être utilisé.

La figure 1 fait apparaitre les différents modules d'EOLSR :

1. le module **Neighborhood Discovery**, chargé de la découverte du voisinage,
2. le module **Strategic Tree Construction**, chargé de construire les arbres routés sur les noeuds stratégiques,
3. le module **Energy Class Determination**, chargé de déterminer à partir de l'énergie résiduelle à quelle classe le noeud appartient à l'instant courant.

Nous détaillons ci-après ces différents modules.

1.3 Module détermination de la classe énergétique

Le module Energy Class Determination est chargé de déterminer à quelle classe énergétique un noeud appartient. Cette appartenance est déterminée par l'énergie résiduelle du

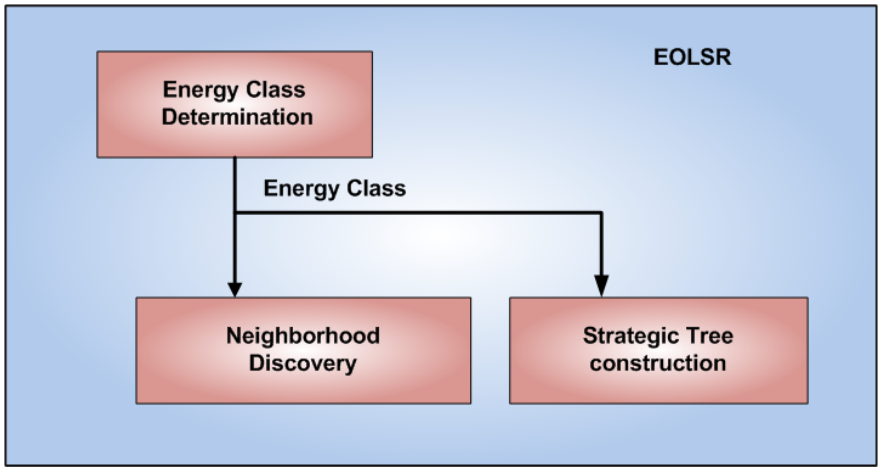


FIGURE 1 – Les différents modules dans EOLSR

noeud. La connaissance de l'énergie résiduelle est obtenue à l'aide d'un modèle énergétique maintenu par ailleurs.

Nous définissons trois classes énergétiques de noeuds qui sont par ordre décroissant :

1. classe High : ce noeud dispose d'une alimentation sur secteur et n'a aucune contrainte énergétique ;
2. classe Medium : ce noeud est alimenté par une batterie et pour l'instant son niveau d'énergie résiduelle est jugé suffisant ;
3. classe Low : ce noeud est alimenté par une batterie et pour l'instant son niveau d'énergie résiduelle est jugé faible.

Chaque noeud est chargé de déterminer la classe énergétique à laquelle il appartient. Pour un noeud qui n'est pas de classe High, la classe à laquelle il appartient peut varier au fil du temps.

Les règles suivantes s'appliquent :

1. Pour entrer dans la classe Medium, un noeud doit avoir un niveau d'énergie résiduelle supérieur ou égal à MIN_ENERGY_HIGH.
2. Pour se maintenir dans la classe Medium, un noeud doit avoir un niveau d'énergie résiduelle supérieur ou égal à MIN_ENERGY_LOW.
3. Nous avons bien évidemment $MIN\_ENERGY\_HIGH > MIN\_ENERGY\_LOW$.

Ce module fournit aux deux modules Neighborhood Discovery et Strategic Tree Construction la classe énergétique du noeud : High, Medium ou Low.

1.4 Module découverte du voisinage dans EOLSR

Dans EOLSR, comme dans OLSR, la découverte du voisinage est effectuée par les Hellos. Les Hellos sont enrichis d'information relative à la classe énergétique des noeuds.

1.4.1 Rôle des Hellos

Dans OLSR, les Hellos servent à :

1. Détecter l'apparition de nouveaux liens, et la signaler,
2. Détecter la rupture de liens existants et la signaler,
3. S'assurer de la symétrie des liens à un saut,
4. Sélectionner uniquement les liens à un saut de bonne qualité,
5. Découvrir les voisins à un saut.

Dans EOLSR, les Hellos permettent en plus d'annoncer la classe énergétique du noeud, calculée en fonction de son énergie résiduelle.

1.4.2 Qualité des liens

Afin d'éviter les liens de mauvaise qualité provoquant de nombreuses pertes et conduisant à une forte instabilité des routes, il est proposé qu'EOLSR utilise les informations fournies par la couche MaCARI pour ne retenir que les liens de bonne qualité dans la construction des routes. Pour cela, deux seuils sont introduits relativement à la qualité des liens :

1. $HYST\_THRESHOLD\_HIGH$ indique la qualité de lien minimum pour pouvoir établir un lien ;
2. $HYST\_THRESHOLD\_LOW$ indique la qualité de lien minimum pour maintenir un lien déjà établi. Si la qualité du lien descend en dessous de ce seuil, le lien est immédiatement rejeté.

Ces deux seuils vérifient bien évidemment :
$HYST\_THRESHOLD\_HIGH > HYST\_THRESHOLD\_LOW$. Ceci donne les deux règles suivantes :

Règle 1 : Un lien n'est jugé de bonne qualité que si la puissance de réception est supérieure à $HYST\_THRESHOLD\_HIGH$;
Règle 2 : Un lien de bonne qualité le reste tant que la puissance de réception est supérieure à $HYST\_THRESHOLD\_LOW$.

1.5 Module construction des arbres vers les noeuds stratégiques

Le module Strategic Tree Construction est chargé de construire un arbre pour chaque noeud stratégique. Chaque noeud maintient une table de topologie, appelée Strategic Tree Table, mémorisant les informations reçues dans les messages STC (Strategic Topology Control) diffusés par les noeuds stratégiques. Cette table permet à tout noeud d'avoir une route vers tout noeud stratégique. Cette construction d'arbre fonctionne pour n'importe quel coût utilisé, sous réserve que le coût de tout lien soit strictement positif. Nous proposons d'utiliser un coût énergétique.

1.5.1 Calcul du coût des routes

L'énergie dissipée lors d'une transmission par le noeud N_i est égale à :

$$Energy(transmission\ par\ N_i) = E_{transmit(i)} + n * E_{receive} \quad (1)$$

avec :

- $E_{transmit}$ l'énergie dissipée en transmission par l'émetteur du message,
- $E_{receive}$ l'énergie dissipée en réception par tout noeud non endormi situé à portée radio de l'émetteur.
- n désignant le nombre de noeuds à distance ≤ 1 saut du noeud émetteur N_i.
  Remarquons que si les noeuds peuvent dormir, il ne faut prendre en compte dans n que les noeuds qui ne dorment pas durant cette transmission.

Pour calculer l'énergie dissipée sur une route par la transmission d'un paquet de sa source à sa destination, nous sommons l'énergie consommée par chacune des transmissions effectuées sur la route empruntée par le paquet. L'énergie dissipée pour acheminer de bout-en-bout un message d'un flux quelconque, dénoté flow, sur une route quelconque est donnée par la formule 2 :

$$energy(flow) = \sum_{N_i \in sender(flow)} (Energy\ (transmission\ par\ N_i)) \quad (2)$$

Dans cette équation, le noeud N_i désigne un noeud émetteur quelconque sur la route empruntée par le flux.

Dans le but de maximiser la durée de vie du réseau, nous imposons les règles suivantes dans le choix des routes, en nous appuyant sur les classes énergétiques des noeuds :

1. ne passer par un noeud de classe énergétique Low que s'il n'existe pas d'autre choix.
2. préférer une route passant par des noeuds de classe High sous réserve que le nombre de sauts ne soit pas prohibitif par rapport à une route utilisant des noeuds Medium.

Pour tenir compte de la classe énergétique de l'émetteur et de ses voisins à un saut, nous définissons trois coefficients : α_{High}, α_{Medium}, α_{Low}, avec $1 \leq \alpha_{High} < \alpha_{Medium} < \alpha_{Low}$.

Nous différencions par classe d'énergie les voisins à un saut de l'émetteur et désignons par :

1. n_1 le nombre de voisins à 1 saut non endormis de classe High de l'émetteur,
2. n_2 le nombre de voisins à 1 saut non endormis de classe Medium de l'émetteur,
3. n_3 le nombre de voisins à 1 saut non endormis de classe Low de l'émetteur.

Remarque : Dans la présérie, nous supposerons que tous les voisins à 1 saut d'un noeud quelconque sont éveillés lorsque ce noeud transmet.

Nous définissons β le rapport :

$$\frac{E_{receive}}{E_{transmit}} = \frac{P_{receive}}{P_{transmit}},\ \text{avec :} \quad (3)$$

1. $P_{transmit}$ la puissance de transmission dissipée par l'émetteur du message,
2. $P_{receive}$ la puissance dissipée en réception par tout noeud non endormi situé à portée radio de l'émetteur.

Pour éviter les calculs avec les flottants, nous prenons :

$$\beta = \frac{P_{receive}}{P_{transmit}} * \lambda, \text{ avec } \lambda \text{ un entier } > 1 \tag{4}$$

.

Nous définissons le coût d'une transmission par un noeud en fonction de sa classe énergétique et de la classe énergétique de ses voisins à un saut, à partir de la formule 1. Nous obtenons :

- dans le cas d'un noeud de classe High :

$$Cost(transmission\ par\ noeud\ High) = \lambda * \alpha_{High} + (n_1 * \alpha_{High} + n_2 * \alpha_{Medium} + n_3 * \alpha_{Low}) * \beta. \tag{5}$$

- dans le cas d'un noeud de classe Medium :

$$Cost(transmission\ par\ noeud\ Medium) = \lambda * \alpha_{Medium} + (n_1 * \alpha_{High} + n_2 * \alpha_{Medium} + n_3 * \alpha_{Low}) * \beta. \tag{6}$$

- dans le cas d'un noeud de classe Low :

$$Cost(transmission\ par\ noeud\ Low) = \lambda * \alpha_{Low} + (n_1 * \alpha_{High} + n_2 * \alpha_{Medium} + n_3 * \alpha_{Low}) * \beta. \tag{7}$$

Le coût d'une route comportant h sauts est obtenu en sommant les coûts de transmission pour chacun des h sauts.

$$Cost(path) = \sum_{i=1}^{h} (Cost(transmission\ by\ the\ i^{th}\ node\ on\ the\ path)) \tag{8}$$

avec h : le nombre de sauts de la route.

La valeur de $Cost(path)$ sera codée sur 2 octets. Ce qui autorise par exemple les possibilités suivantes, avec $\lambda = 4$ et $\alpha_{Medium} = 4$:

- au maximum 64 voisins Medium par noeud et une route comportant jusqu'à 63 sauts ;
- au maximum 128 voisins Medium par noeud et une route comportant jusqu'à 31 sauts ;
- ou au maximum 256 voisins Medium par noeud et une route comportant jusqu'à 15 sauts.

1.5.2 Sélection des routes

Au lieu de prendre uniquement le nombre de sauts comme critère de choix de la meilleure route, nous utilisons un critère de coût. Un coût énergétique est calculé pour chaque route

(voir équation 8), comme cela a été explicité dans la section précédente. Ce coût prend en compte l'énergie dissipée en transmission par l'émetteur et en réception par tous les noeuds qui vont écouter cette transmission, ainsi que le niveau d'énergie résiduelle des noeuds visités. Ce niveau est représenté par une classe énergétique.

L'algorithme de routage EOLSR consiste à sélectionner la route de moindre coût. Remarquons que s'il existe plusieurs routes de moindre coût, un choix arbitraire est effectué : la route avec la plus petite adresse de prochain saut (next hop) est retenue.

1.5.3 Rôle des STC

Dans EOLSR, les STC permettent à chaque noeud de calculer une route vers tout noeud stratégique, en évitant les noeuds de moindre énergie résiduelle et en sélectionnant les routes de moindre coût énergétique. Chaque noeud conserve ainsi comme parent le noeud voisin symétrique lui permettant de minimiser le coût énergétique pour atteindre le noeud stratégique.

Les messages STCs sont générés périodiquement par tout noeud stratégique et propagés à travers tout le réseau. Un message STC n'est propagé que s'il a permis la mise à jour de la table STT, Strategic Tree Table. En d'autres termes, ce message STC a permis d'améliorer la route vers le noeud stratégique ou plus simplement d'entretenir cette route en prolongeant la durée de validité du parent dans l'arbre. Le parent dans l'arbre est le prochain saut vers le noeud stratégique racine de cet arbre.

Par définition, **le parent d'un noeud quelconque est dit valide** tant que :
- le lien entre ce noeud et ce parent est symétrique,
- le tuple associé à ce parent dans la table STT n'a pas expiré.

De même, par définition, pour tout noeud, **un fils est dit valide** tant que :
- le lien entre ce noeud et ce fils est symétrique,
- le tuple associé à ce fils dans la table des fils n'a pas expiré.

1.5.4 Rôle des Tree Status

Les messages Tree Status ne sont utilisés que pour les arbres devant être coloriés. Ils permettent d'indiquer au noeud stratégique l'état de l'arbre qu'il construit (les changements de topologie qu'il y a eu telle que l'apparition d'un noeud). En particulier, l'état de l'arbre inclut la stabilité de l'arbre. En effet, seuls les arbres devant être coloriés qui sont stables peuvent être coloriés.
De plus, la priorité d'un noeud dans l'algorithme de coloriage (c'est à dire son ordre de coloriage par rapport aux noeuds qui ne doivent pas avoir la même couleur que lui) qui est donnée par le nombre de descendants du noeud dans l'abre, est échangée dans ce message. Le message Tree Status contient un champ *flags* dont les flags 'stable' et 'unstable'. Dans la suite, la notation *Tree Status (stable)* (respectivement *unstable*) veut dire que le flag *stable* (respectivement *unstable*) envoyé dans ce message est mis à 1.

Un arbre est identifié de manière unique par le couple ($strategic\_node, tree\_seqnum$), c'est-à-dire adresse du noeud stratégique racine de l'arbre et numéro de séquence de cet arbre.

Un noeud émet un message Tree Status (stable) si le sous arbre dont il est racine est stable. Plus précisément, tout noeud surveille **une condition de stabilité locale** pour un arbre de numéro de séquence et de racine donnés. Cette condition est donnée par :

1. avoir un père dans l'arbre de routage,
2. ne pas avoir enregistré un changement de voisinage (apparition/disparition) de voisins pendant un intervalle de temps prédéfini.

Tout noeud qui a détecté cette condition de stabilité, et qui a reçu un message Tree Status(stable) de chacun des ses fils pour ce numéro de séquence d'arbre envoie à son tour ce message Tree Status (stable) pour cet arbre. Dans ce cas, le message Tree Status contient le nombre de descendants du noeud émetteur du message. Lorsque le noeud stratégique a reçu un message Tree Status (stable) de chacun de ses fils pour le numéro de séquence de l'arbre courant, il fait passer l'état de l'arbre à stable. L'arbre peut alors être colorié. Le couple (nombre de descendants d'un noeud, adresse du noeud) est utilisé comme priorité du noeud dans l'algorithme de coloriage.

La réception du message Tree Status(unstable) permet au noeud stratégique racine de l'arbre de savoir que l'arbre de numéro de séquence indiqué est instable. Le reste des flags indique également les changements de topologie dans cet. La réaction vis-à-vis de ces changements est une décision centralisée au niveau de la racine de l'arbre. Par exemple, quand le coloriage n'est pas encoré déclenché, la racine peut incrémenter le numéro de séquence de l'arbre, un nouvel arbre doit être construit et stabilisé.

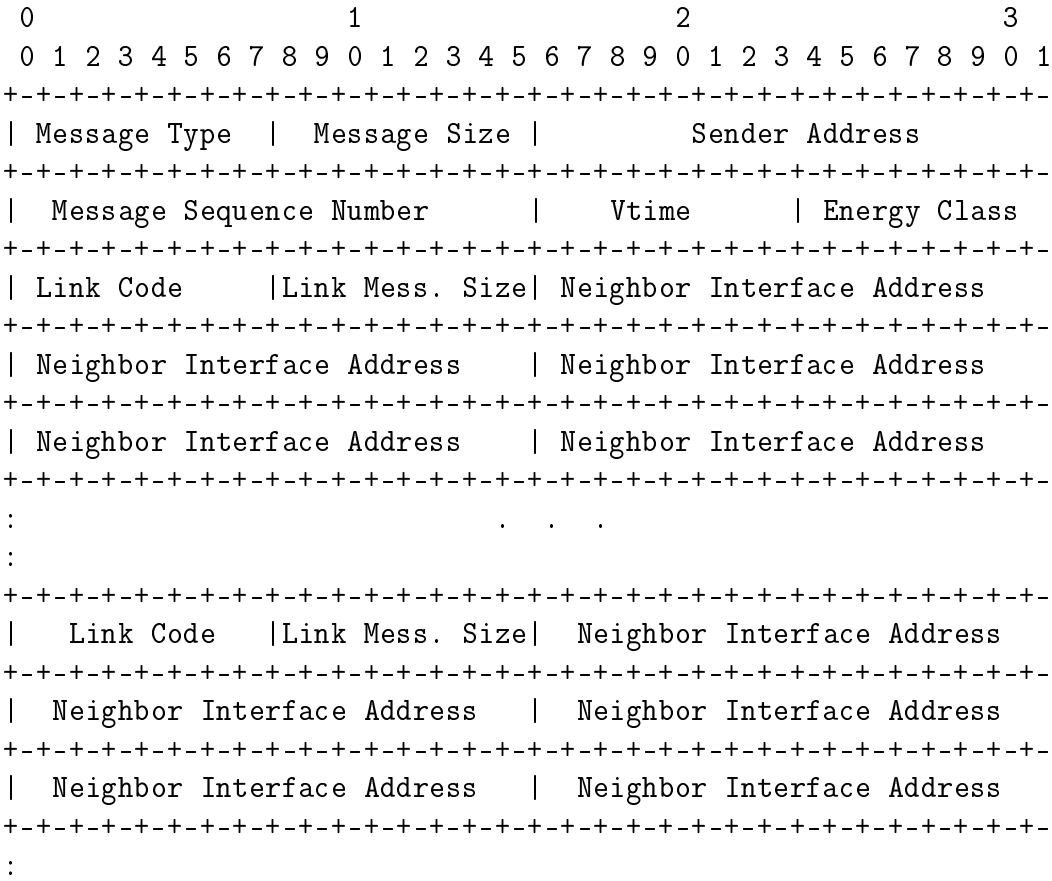
Nous détaillerons dans la section 4 les règles de génération et de traitement des messages Hello, STC et Tree Status.

2 Format des messages EOLSR

EOLSR utilise les messages suivants :
- Hello pour la découverte du voisinage,
- STC pour l'obtention d'une route vers un noeud stratégique,
- Tree Status pour notifier le noeud stratégique du statut de l'arbre construit. Un arbre qui doit être colorié ne le sera que si son statut est stable.

2.1 Format d'un message Hello

Tout message Hello comporte les champs suivants :



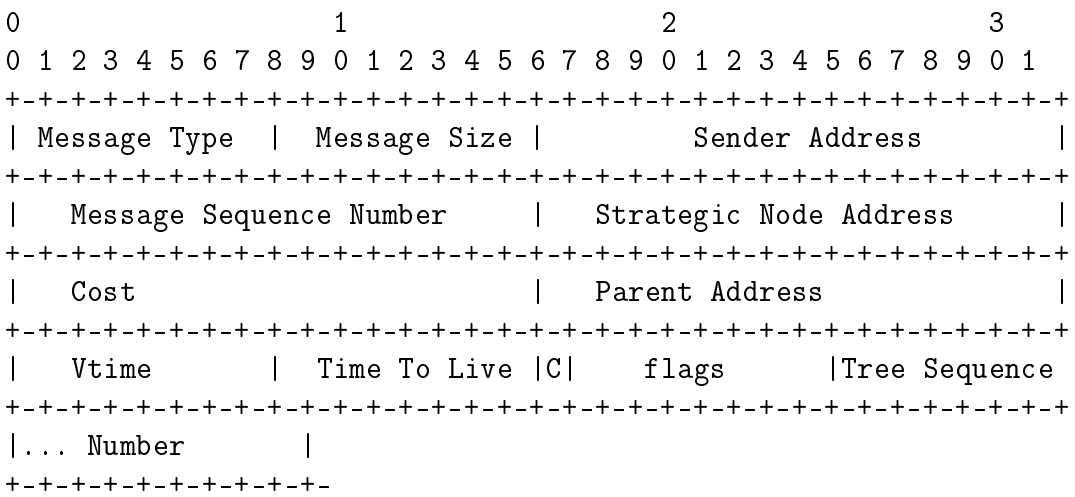
- Message Type : indique le type de message (Hello dans le cas présent) : 1 octet
- Message Size : ce champ indique la taille du message en octets : 1 octet
- Sender Address : l'adresse du noeud qui a généré le message : 2 octets.
- Message Sequence Number : le numéro de séquence du message. Quand un noeud génère un nouveau message, il lui attribue un identificateur unique. Ce numéro est incrémenté par 1 pour chaque nouveau message généré par ce noeud : 1 octet.
- Vtime : ce champ indique le temps pendant lequel le récepteur doit considérer les informations contenues dans ce message comme valides : 1 octet.
- Energy Class : ce champ désigne la classe énergétique du noeud : 1 octet.
- Link Code : Ce champ indique le type de lien entre le noeud source et chacun de ses voisins à 1 saut : SYM, ASYM : 1 octet
- Link Message Size : ce champ indique la taille en octet de la partie du message commençant par le champ Link Code jusqu'au Link Code suivant : 1 octet.
- Pour chaque voisin ayant ce type, ajouter Neighbor Interface Address : l'adresse du noeud voisin ayant ce type de lien : 2 octets.

– { Link Code : 1octet,
Link Message Size : 1 octet,
Pour tout voisin avec ce type de lien : Node Interface Address : 2 octets.}*

Remarque : Dans la présérie, les messages Hello ne peuvent pas être fragmentés.

2.2 Format d'un message STC

Le message STC comprend les champs suivants :



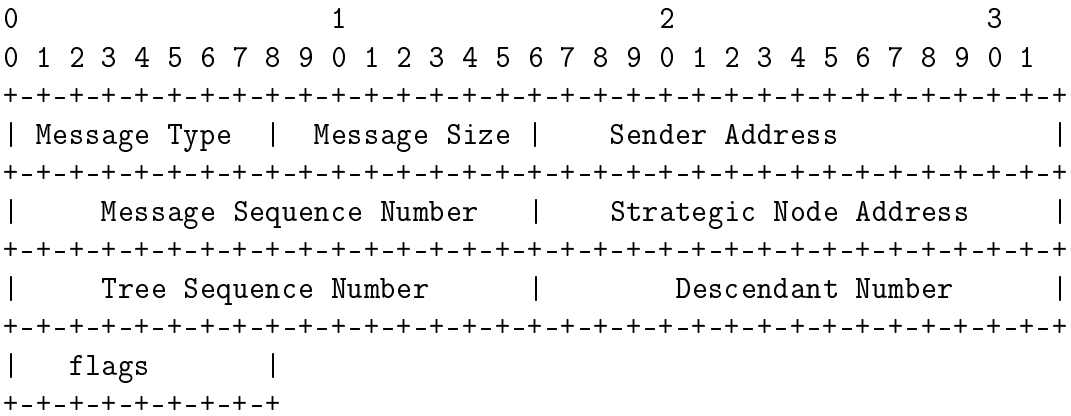
- Message Type : indique le type de message (STC dans le cas présent) : 1 octet
- Message Size : ce champ indique la taille du message en octets : 1 octet
- Sender Address : adresse du noeud émetteur du STC : 2 octets.
- Message Sequence Number : le numéro de séquence du message STC. Le noeud stratégique incrémente de 1 ce numéro de séquence pour chaque nouveau message STC qu'il génère : 2 octets.
- Strategic Node Address : l'adresse du noeud stratégique qui a généré le message et vers lequel chaque noeud construit une route : 2 octets.
- Cost : ce champ désigne le coût énergétique de la transmission de bout en bout. Ce coût est initialisé par le noeud stratégique et est mis à jour à chaque retransmission. Il inclut le coût d'émission par l'émetteur de ce message et le coût de réception par ses voisins à 1 saut non endormis : 2 octets.
- Parent Address : c'est l'adresse du père de l'émetteur dans l'arbre. Le noeud stratégique initialise ce champ à 0 : 2 octets.
- Vtime : ce champ indique le temps pendant lequel le récepteur doit considérer les informations contenues dans ce message comme valides : 1 octet.
- Time To Live : ce champ indique le nombre maximum de sauts que ce message peut subir. A chaque réception de ce message ce nombre doit être décrémenté par 1. Si le time to live (TTL) atteint 0, ce message ne peut plus être propagé : 1 octet.
- C : vaut 1 si l'arbre doit être colorié et 0 sinon : 1 bit.
- flags : indique l'état de l'arbre : 1 octet dont les bits suivants :

- FLAG_INCONSISTENT_STABILITY : vaut 1 s'il y a une incohérence au niveau de la stabilité (noeud non stable alors que la racine est stable),
- FLAG_STABLE : vaut 1 si la condition de stabilité est valide,
- FLAG_UNSTABLE : vaut 1 si la condition de stabilité n'est pas valide,
- FLAG_TREE_CHANGE : vaut 1 s'il y a un changement des liens dans l'arbre (changement au niveau du père ou d'un fils),
- FLAG_NEW_NEIGHBOR : vaut 1 si le noeud découvre un nouveau lien symétrique.

– Tree Sequence Number : n'est pas significatif si C=0 ;
sinon le numéro de séquence de l'arbre généré par ce STC et dont la racine est le noeud stratégique. Initialement à 0, il est incrémenté de 1 par le noeud stratégique sur réception d'un Tree Status (unstable) de l'arbre courant : 2 octets.

2.3 Format d'un message Tree Status

Le message Tree Status comprend les champs suivants :



- Message Type : indique le type de message (Tree Status dans le cas présent) : 1 octet
- Message Size : ce champ indique la taille du message en octets : 1 octet
- Sender Address : c'est l'adresse du noeud émetteur du Tree Status sur 2 octets.
- Message Sequence Number : le numéro de séquence du message STC auquel ce message tree Status répond : 2 octets.
- Strategic Node Address : l'adresse du noeud stratégique qui est racine de l'arbre : 2 octets.
- Tree Sequence Number : le numéro de séquence de l'arbre pour lequel le noeud émetteur donne son statut : 2 octets.
- Descendant Number : le nombre de descendants du noeud émetteur du message Tree Status. Cette information n'est valide que si le status est stable : 2 octets.
- flags : c'est le même octet qui est envoyé dans le message STC.

2.4 Périodicité des messages

Déterminer la périodicité des messages de contrôle d'un protocole de routage nous confronte à deux écueils :

1. Le réseau considéré disposant d'une bande passante limitée, la périodicité des messages devra être allongée en conséquence ;
2. Une période allongée conduit à une faible réactivité du protocole et augmente la latence de détection et donc de recouvrement des changements de topologie.

Il faut donc trouver le meilleur compromis.

2.4.1 Périodicité des messages Hello

Nous proposons de déterminer deux périodes de Hello :

1. **HelloMax** : qui représente la période d'envoi des Hellos dans un réseau stable. Cette période est égale à l'intervalle de temps entre deux demandes d'envoi successives d'un Hello par un noeud quelconque du réseau. C'est la valeur maximum et la valeur par défaut, vers laquelle tend à revenir le réseau.
2. **HelloMin** : qui représente l'intervalle de temps minimum écoulé depuis la demande d'envoi du dernier Hello par un noeud ayant détecté des changements de topologie. Remarquons qu'attendre HelloMin permet de grouper les changements détectés en un seul message.

En conclusion, nous distinguons deux cas :

1. en période stable, chaque noeud émet un Hello tous les HelloMax ;
2. en cas de changement du voisinage à un saut, chaque noeud détectant la rupture d'un lien avec un voisin à un saut ou l'apparition d'un nouveau lien attend au pire HelloMin pour diffuser son Hello.

Remarque : Dans la présérie, nous aurons HelloMin = HelloMax.

2.4.2 Périodicité des messages STC

Les messages STC permettent aux noeuds de construire une route vers le noeud stratégique à l'origine du message STC. Comme pour les Hellos, nous distinguons deux périodes d'envoi des STCs :

- **STCMax** qui représente la période d'envoi des STCs dans un réseau stable. Cette période est égale à l'intervalle de temps entre deux demandes d'envoi successives d'un STC par un noeud stratégique. C'est la valeur maximum et la valeur par défaut vers laquelle tend à revenir le réseau.
- **STCMin** qui représente l'intervalle de temps minimum écoulé depuis la demande d'envoi du dernier STC par un noeud stratégique ayant détecté des changements de topologie.
  Remarquons qu'attendre STCMax permet de grouper les changements détectés en un seul message.

Remarque : Dans la présérie, nous aurons STCMin = STCMax.

Nous détaillerons dans la section 4 les règles de génération et de traitement des messages Hello, STC et Tree Status.

3 Bases d'information maintenues

Grâce à l'échange des messages de contrôle dans EOLSR, chaque noeud collecte des informations sur le réseau et les stocke dans des bases d'information. Dans cette partie, nous décrivons les informations stockées au niveau de chaque noeud. Un noeud détient une base d'information :

- relative à lui même, notée *base_state*,
- relative à son voisinage, notée *eond_state*, et qui est construite via l'échange des messages Hellos,
- relative à l'arbre de routage, notée *eostc_state*, et qui est construite via l'échange des messages STC.

3.1 Informations relatives au noeud lui-même : *base_state*

Un noeud maintient les paramètres suivants :

- thisnode_class : codé sur un octet et vaut : High si ce noeud est sur secteur, sinon Medium ou Low selon son énergie résiduelle,
- thisnode_address : l'adresse du noeud sur 2 octets.

3.2 Détection des voisins : base d'information de voisinage : *eond_state*

La base d'information de voisinage contient des informations sur le voisinage :

- neighbor_table : la table des voisins,
- MAX_NEIGHBOR : le nombre maximum des voisins,
- hello_seq_num : le numéro de séquence du dernier message Hello envoyé,
- has_neighborhood_changed : un booléen qui vaut TRUE si le voisinage a changé.

Tout tuple de la table de voisins neighbor_table contient :

- N_neighbor_addr : est l'adresse du voisin : 2 octets.
- N_status : indique si le voisin est symétrique ou non : 1 bit.
- N_SYM_time : représente l'instant jusqu'auquel on considère ce noeud comme voisin symétrique : 2 octets.
- N_ASYM_time : représente l'instant jusqu'auquel on considère ce noeud comme entendu : 2 octets.
- N_neighbor_nb : indique le nombre de voisins symétriques de ce voisin : 1 octet.
- N_class : indique la classe de ce voisin : 1 octet.

Les règles suivantes s'appliquent :
- Tant que N_SYM_time n'est pas expiré, le lien avec ce noeud est déclaré symétrique.
- Si N_SYM_time expire, le lien avec ce noeud est déclaré asymétrique.

3.3 Base d'information relative à l'arbre de routage : *eostc_state*

Cette base d'information est construite via l'échange des messages STC, et pointe également vers les deux bases *eond_state*, *base_state*. Elle contient :
- STT : (Strategic Tree Table) Cette table contient pour chaque noeud stratégique un tuple contenant les informations nécessaires à la construction/maintenance de l'arbre dont la racine est ce noeud stratégique.
- MAX_STC_TREE : nombre maximum de tuples dans la table STT,
- S_color, 1 bit qui vaut 1 si l'arbre doit être colorié et vaut 0 sinon,
- **serena_info** : pour les arbres devant être coloriés, tout noeud maintient, en plus, cette table contenant des informations relatives au coloriage de l'arbre concerné,
- MAX_STC_SERENA_TREE : nombre maximum des arbres à colorier, c'est à dire la taille maximale de serena_info.

Tout tuple de la table STT contient les champs suivants :
S_strategic_addr S_validity_time S_parent_addr S_cost S_seqnum S_color.
- S_strategic_addr : Dans l'arbre dont le noeud stratégique d'adresse S_strategic_addr (2octets) est la racine, le noeud considéré utilise le noeud d'adresse S_parent_addr (2octets) comme père pour atteindre la racine.
- S_seqnum : 2 octets, est le numéro de séquence du STC duquel provient cette information.
- S_cost : Le coût de la route vers le noeud stratégique correspondant (2 octets).
- S_validity_time : Les informations contenues dans ce tuple sont valides jusqu'à S_validity_time, 2 octets.

La table serena_info n'est maintenue que pour un arbre devant être colorié, elle contient :
- S_tree_seqnum : 2 octets, est le numéro de séquence de l'arbre en cours de construction.
- S_is_subtree_stable : 1 bit qui vaut 1 si le noeud considère le sous arbre dont il est racine stable, et vaut 0 sinon,
- is_neighborhood_stable : vaut 1 si le parent, les voisins, et les enfants sont stables, et 0 sinon.
- has_sent_stable : vaut 1 si le noeud a envoyé un message Tree Status avec comme status stable, et 0 autrement,
- should_generate_tree_status : vaut 1 si le noeud peut envoyer un message Tree Status, et vaut 0 sinon,
- S_flags : ce sont les flags échangés dand les messages STC, et les messages Tree Status.

- **S_childrenset** : mémorise les informations relatives aux enfants du noeud considéré dans cet arbre. Avec pour chaque enfant, un tuple **(C_child_addr, C_validity_time, C_tree_status, C_descendant_number)** contenant les champs suivants :
  - C_child_addr : l'adresse du noeud child : 2 octets
  - C_validity_time : la date limite de validité des informations contenues dans ce tuple
  - C_tree_status : le statut de l'arbre dont le noeud C_child_addr est racine : 1 bit
  - C_descendant_number : le nombre de noeuds dans l'arbre dont C_child_addr est racine : 2 octets.

Si l'option backup est activée, chaque tuple de la table STT possède les champs additionnels suivants : **S_backup_validity_time S_backup_parent_addr S_backup_cost S_ backup_ seqnum**

- Ces champs additionnels contiennent des informations dont la validité est limitée par S_backup_validity_time : 2 octets.
- Le champ S_backup_parent_addr contient l'adresse du backup du père dans l'arbre : 2 octets.
- Le coût de cette route est S_backup_cost : 2 octets.
- S_backup_seqnum est le numéro de séquence du STC duquel provient cette information : 2 octets.

4 Génération et traitement des messages Hello, STC et Tree Status

Dans ce qui suit, nous allons présenter la génération et le traitement des messages de contrôle dans EOLSR à savoir les messges Hellos, les messages STCs et les messages Tree Status.

4.1 Message Hello

L'échange périodique des messages Hellos est le mécanisme utilisé dans EOLSR (comme dans OLSR) pour qu'un noeud connaisse ses voisins à 1 saut ainsi que le type des liens avec ces voisins. Dans cette partie, nous allons décrire la génération et la réception d'un message Hello.

4.1.1 Génération d'un message Hello

Les messages Hellos permettent de :

- Détecter les voisins à 1 saut,
- Connaître le type des liens avec ses voisins.

En d'autres termes, ces messages permettent de découvrir la topologie locale autour d'un noeud. Chaque noeud envoie périodiquement des messages Hellos :

- chaque HelloMin tant que le voisinage n'est pas stable.

– chaque HelloMax sinon.

Remarque : Dans la présérie, nous aurons HelloMin = HelloMax.
Le message Hello est généré à partir de la base d'informations stockées sur le voisinage du noeud.

Pour générer un message Hello, il faut remplir chaque champ dans le message comme suit :

- Message Type = Hello
- Message Size = taille du message en octets
- Sender Address = thisnode_address
- Message Sequence Number = numéro de séquence courant incrémenté de 1
- Vtime = la constante NEIGH_HOLD_TIME.
- Calculer la classe énergétique du noeud dans thisnode_class
  - Si le noeud est sur secteur alors thisnode_class = High
  - Sinon thisnode_class = Medium ou Low selon le niveau d'énergie résiduelle
  - Energy Class = thisnode_class
- Pour chaque tuple de voisins (N_neighbor_addr, N_status, N_SYM_time, N_ASYM_time, N_neighbor_nb, N_class) dans la table de voisinage, il faut le publier dans le message Hello avec Link Code déterminé comme suit :
  - Si N_SYM_time >= temps courant (non expiré) alors Link Code = SYM
  - Sinon, si N_ASYM_time >= temps courant (non expiré) et N_SYM_time < temps courant (expiré) alors Link Code = ASYM

Les règles suivantes s'appliquent à la génération des Hellos :

- Seuls les Link Codes pour lesquels existe au moins un noeud sont émis.
- Le Link Message Size associé au Link Code comptabilise le nombre d'octets utilisés depuis ce Link Code jusqu'au prochain Link Code ou à la fin du message. Il est donc égal à 2+ (nombre de voisins de ce type *2).
- Le message Hello ainsi généré est diffusé à un saut.

4.1.2 Réception d'un message Hello

Un message Hello n'est jamais forwardé. A la réception d'un message Hello, un noeud met à jour les tuples des voisins comme suit :

1. Recherche de l'entrée correspondante dans la table neighbor_table :
   - S'il n'existe pas un tuple de voisin tel que :
     N_neighbor_addr == Sender Address, alors :
     - Si la puissance fournie par MaCARI pour ce Hello reçu < HYST_THRESHOLD_HIGH alors ignorer le message reçu, exit.
     - sinon un nouveau tuple est créé avec :
       - N_neighbor_addr = Sender Address,
       - N_SYM_time = temps courant -1 (expiré),
       - N_ASYM_time = temps courant -1 (expiré),

- Le noeud met à jour la variable has_neighborhood_changed = True (apparition d'un voisin).

2. Mise à jour du tuple (nouveau ou existe déjà) qui satisfait N_neighbor_addr = Sender Address avec :
   - Si la puissance fournie par MaCARI pour ce Hello reçu < HYST_THRESHOLD_LOW alors :
     - ignorer le message reçu,
     - supprimer ce tuple de voisin,
     - Le noeud met à jour la variable has_neighborhood_changed = True (disparition d'un voisin).
     - exit.
   - Sinon, modifier le tuple de voisin comme suit :
     - N_ASYM_time = temps courant + Vtime,
     - Si le noeud courant trouve son adresse parmi les adresses listées dans le message alors le tuple est modifié comme suit :
       - N_SYM_time = temps courant + Vtime,
     - N_neighbor_nb = nombre de noeuds publiés dans le Hello reçu avec comme Link Code = SYM
     - N_class = Energy Class reçue dans le Hello
   - Mettre à jour le status du voisin de telle sorte que :
     - si le voisin correspond à un lien symétrique alors N_status = SYM.
     - sinon N_status = ASYM.
3. S'il y a eu un changement de **status** du voisin mis à jour,
   alors faire has_neighborhood_changed = True.

4.2 Message STC

Les messages STC permettent de construire l'arbre dont la racine est le noeud stratégique générateur du message STC. Un arbre est identifié de manière unique par le couple (strategic_node, tree_seqnum), c'est-à-dire par l'adresse du noeud stratégique racine de l'arbre et par le numéro de séquence de cet arbre. Chaque noeud qui n'est pas le noeud racine dispose d'un parent dans l'arbre qui est le prochain saut vers la racine de l'arbre. Ainsi, les messages STC permettent à chaque noeud de construire une route vers chaque noeud stratégique.

Les routes construites doivent être de moindre coût énergétique. De plus, elles doivent éviter les noeuds avec faible énergie résiduelle et partager la charge entre les noeuds du réseau pour éviter que certains noeuds épuisent leurs batteries plus vite que les autres. La métrique utilisée pour construire les routes est le coût énergétique.

4.2.1 Génération d'un message STC

Le message STC n'est généré que par les noeuds stratégiques. Les STCs sont envoyés périodiquement :

- chaque STCMin, tant que l'arbre n'est pas stable,
- chaque STCMax sinon.

Remarque : Dans la présérie, nous aurons STCMin = STCMax.

Pour générer un message STC, tout noeud utilise les informations stockées dans la base *eostc_state*, et *serena_info* pour remplir chaque champ dans le message comme suit :

- Message Type = STC
- Message Size = taille du message en octets
- Sender Address = thisnode_address
- Message Sequence Number = numéro de séquence courant incrémenté de 1 à chaque génération de STC, initialement vaut 0.
- Strategic Node Address = thisnode_address
- Cost = 0 si on considère le coût de la route vers le puits.
- Parent Address = 0 pour le noeud stratégique racine de l'arbre.
- Vtime = TREE_HOLD_TIME.
- Time To Live = MAX_HOP
- Flag C = 1 si cet arbre doit être colorié et 0 sinon
- Le champ flag vaut 0 si C = 0, sinon ajouter le champ flag (voir section 2.2) selon l'état du noeud et les changements de topologie enregistrés,
- Tree Sequence Number vaut 0 si C=0, sinon est initialisé à 0 lors de l'envoi du premier STC. Ce numéro de séquence est incrémenté par le noeud stratégique après réception par ce noeud d'un message Tree Status (unstable) pour l'arbre de numéro de séquence courant. Autrement dit, ce paramètre indique le numéro de coloriage de l'arbre, si le coloriage est relancé, ce numéro est incrémenté.

4.3 Réception d'un message STC

On appelle "Sender Address" l'adresse du noeud qui a forwardé ce message.

Changement de père sur réception d'un STC : Un noeud ayant un père valide ne change de père qu'au profit d'un nouveau père :

- de plus petit coût
- et de numéro de séquence
  - supérieur au numéro de séquence du père courant,
  - ou (égal au numéro de séquence du père courant et ce noeud n'est pas fils du noeud considéré).

Propagation du STC reçu : Un noeud qui :
- change de père
- ou met à jour le numéro de séquence ou le coût de son père doit propager le STC reçu.

Lorsque l'option backup est supportée, elle permet au noeud d'avoir un backup qui pourra être utilisé en cas de défaillance du père.
- Dans la négative, le noeud est obligé d'attendre la prochaine réception de STC pour retrouver un père. La latence est alors au pire d'une période de STC en l'absence de perte de message.
- Si par contre, le noeud avait stocké un backup et que celui-ci est toujours valide, il le prend comme père dès lors que son père devient invalide.
- Le numéro de séquence du STC est attribué par le noeud stratégique lors de sa génération. Toutefois, un noeud non stratégique peut être amené à modifier le numéro de séquence du STC propagé lorsque l'option backup est supportée et ce noeud prend le backup comme père. Il utilisera alors le maximum entre le numéro de séquence mémorisé pour le backup et le numéro de séquence mémorisé pour le père.

Si un arbre doit être colorié et uniquement dans ce cas,
- Un noeud maintient la connaissance de ses enfants et les stocke dans S_childrenset. Un noeud détecte qu'un autre noeud est son enfant dès lors qu'il reçoit un STC de ce noeud avec Parent Address == son adresse thisnode_addr.
- Pour un arbre devant être colorié, un noeud informe la racine de l'arbre de l'état de l'arbre (stable, instable, changements de topologie, etc). Par conséquent, tout noeud maintient et échange des flags qui décrivent la stabilité de l'arbre et les changements de topologie que le noeud a enregistré.

A la réception d'un message STC, le noeud commence par identifier l'émetteur du message et la racine Strategic Node Address publiés dans le message. Selon ces informations, le noeud est dans l'un des 4 cas suivants :

1. Si l'expéditeur de ce message n'appartient pas à l'ensemble des voisins symétriques alors le message doit être supprimé, exit.
2. Si l'adresse de la racine (Strategic Node Address) envoyée dans le message correspond à une nouvelle racine : c'est à dire, s'il n'existe pas un tuple de noeud stratégique dans la table STT tel que : S_strategic_addr == Strategic Node Address reçu dans le message STC, alors appeler la procédure 4.3.1
3. Si l'expéditeur du message est un fils (il existe un tuple dans la table S_Childrenset, tel que C_child_addr = Sender Address) et que le récepteur est la racine de l'arbre (thisnode_address = Strategic Node Address), alors exécuter la procédure 4.3.2
4. Sinon, il existe un tuple pour cette racine dans la table STT, exécuter la procédure 4.3.3 de mise à jour de ce tuple.

4.3.1 Découverte d'un nouvel arbre

Cette procédure est appelée par un noeud qui reçoit un message STC noté *message* avec une adresse Strategic node Address qui n'existe pas dans la table STT. Dans ce cas, si le noeud n'a pas atteint le nombre maximum MAX_STC_TREE d'arbres dans sa table, il ajoute un tuple *tree* dans sa table STT avec les informations suivantes :

- tree.S_Strategic_address = message.Strategic Node Address
- tree.S_parent_address = message.Sender Address
- ajouter les autres champs relatifs à cet arbre en appelant la procédure refresh_tree(message, tree)
- si (tree.serena_info.is_subtree_stable == True) alors
  tree.serena_info.should_generate_tree_status = True,
- tree.serena_info.is_neighborhood_stable = False,
- tree.serena_info.is_subtree_stable = False,
- tree.serena_info.stability_time = current_time + STABILITY_TIME.

4.3.2 Réception d'un message STC d'un fils

Cette procédure est appelée par un noeud qui reçoit un message STC provenant d'un noeud qui l'a choisit comme père dans l'arbre qu'on note *tree*. Dans ce cas, le noeud recherche dans sa table S_childrenset s'il existe un tuple avec C_child_addr == Sender Address, le met à jour s'il existe, et l'ajoute sinon.

- si le numéro de séquence du STC reçu est $\leq$ tree.S_seqnum stocké alors :
  1. si le noeud Sender Address n'appartient pas à la table S_childrenset alors créer un nouveau tuple avec les informations suivantes :
     - C_tree_status = unstable
     - C_descendant_number = 0
     - si (tree.serena_info.is_subtree_stable == True) alors
       tree.serena_info.should_generate_tree_status = True,
     - tree.serena_info.is_neighborhood_stable = False,
     - tree.serena_info.is_subtree_stable = False,
     - tree.serena_info.stability_time = current_time + STABILITY_TIME
  2. Dans les deux cas, que le fils existe dans la table des fils ou non,
     faire C_validity_time = TREE_HOLD_TIME + temps courant.
- Sinon ignorer. /*ce cas ne doit pas arriver dans le fonctionnement normal du protocole*/

4.3.3 Réception d'un message STC avec une racine connue

Cette procédure est exécutée si le noeud récepteur reçoit un message STC d'un voisin symétrique. De plus, ce message correspond à un arbre *tree* dont la racine est connue par le récepteur. La première instruction que le récepteur exécute est la vérification de la validité

de son père courant relativement à cet arbre. Deux cas sont possibles :

Premier cas : le père stocké localement est valide en tant que voisin symétrique

1. Si (Sender Address == tree.S_parent_addr) /*l'émetteur est un père*/
   - Si le numéro de séquence du STC reçu dans le message < tree.S_seqnum, alors ignorer le message.
   - sinon, mettre à jour le tuple correspondant en appelant la procédure **refresh_tree(message STC, tree)**. /*l'égalité des numéros de séquence correspond à un changement de l'arbre en amont */

2. l'émetteur est un fils, alors exécuter la procédure 4.3.2.

3. l'émetteur est ni un père ni un fils.
   (a) Vérification si le noeud émetteur était un fils :
   Si le numéro de séquence du STC reçu dans le message est ≥ tree.S_seqnum :
   s'il existe un tuple dans la table des fils tel que (C_child_addr == Sender address) alors effacer ce tuple de cette table.
   (b) Si le (Cost reçu < tree.S_cost)
      i. Si (seq num reçu ≥ tree.S_seqnum) alors :
         - Si option backup et père courant valide et tree.S_cost < tree.S_backup_cost Alors prendre le père courant comme backup :
           - tree.S_backup_validity_time = tree.S_validity_time
           - tree.S_backup_parent_addr = tree.S_parent_addr
           - tree.S_backup_cost = tree.S_cost
           - tree.S_backup_seqnum = tree.S_seqnum
         - tree.S_parent = Sender Address /*changement du père*/
         - si (tree.serena_info.is_subtree_stable == True) alors tree.serena_info.should_generate_tree_status = True,
         - tree.serena_info.is_neighborhood_stable = False,
         - tree.serena_info.is_subtree_stable = False,
         - tree.serena_info.stability_time = current_time + TREE_STABILITY_TIME
         - mettre à jour le tuple *tree* en appelant la procédure refresh_tree (message, tree)
         - générer et propager le STC reçu comme dans la procédure 4.2.1

   (c) Si option backup supportée et Cost reçu < tree.S_cost et Cost reçu < tree.S_backup_cost et Message Sequence Number > tree.S_backup_seqnum Alors prendre le noeud émetteur comme backup :
   - tree.S_backup_validity_time = temps courant + TREE_HOLD_TIME

- tree.S_backup_parent_addr = Sender Address reçu
- tree.S_backup_cost = Cost reçu
- tree.S_backup_seqnum = Message Sequence Number reçu

Deuxième cas : le père du récepteur n'est pas valide en tant que voisin symétrique
Dans ce cas, le voisinage du noeud n'est pas stable. Ce noeud essaie de changer le père courant en considérant l'émetteur du message STC.

1. tree.serena_info.is_neighborhood_stable = False,
2. tree.serena_info.is_subtree_stable = False,
3. tree.serena_info.stability_time = current_time + TREE_STABILITY_TIME

4. si (Message sequence Number reçu $\geq$ tree.S_seqnum) alors :
   - si l'émetteur Sender address n'est pas un fils alors
   - prendre l'émetteur comme père : tree.S_parent = Sender address
   - mettre à jour le tuple correspondant : refrech_tree (message, tree)
   - sinon ignorer le message. /*un message d'un fils avec un numéro de séquence de STC plus grand*/
5. sinon ignorer le message. /*un message avec un numéro de séquence obsolète*/

4.3.4 Procédure refresh_tree

Cette procédure est appelée à la réception d'un message STC, noté *message*, pour mette à jour un tuple *tree* de la table STT comme suit :

- étendre le temps de validité de l'arbre : tree.S_validity_time = temps courant + TREE_HOLD_TIME.
- tree.S_cost = message.Cost
- tree.S_seqnum = le Message Sequence Number reçu
- Si (message.C == 1) (arbre à colorier)
  alors mettre à jour les informations utiles à SERENA (serena_info) et au traitement/émission des messages Tree Status comme suit :
  - Si (le numéro de séquence de l'arbre reçu dans le message $\geq$ tree.serena_info.S_tree_seqnum) alors
    - Arrêtr le processus de coloriage SERENA /*un autre coloriage en cours*/
    - tree.serena_info.S_stability_time = temps courant + TREE_STABILITY_TIME.
  - Sinon, si (message.tree_seqnum == tree.serena_info.S_tree_seq_num) alors
    - mettre à jour S_flags : tree.serena_info.S_flags = (tree.serena_info.S_flags || message.flags)
    - tree.serena_info.should_generate_tree_status = (tree.serena_info.S_flags & message.flags)
  - tree.serena_info.tree_seqnum = message.tree_seqnum
- si (message.ttl != 0) alors tree.ttl = tree.ttl-1

4.4 Messages Tree Status

Les messages Tree Status ne sont utilisés que pour les arbres devant être coloriés. Ils permettent à la racine de l'arbre de connaitre l'état de l'arbre, en particulier savoir si l'arbre construit est stable ou non. Un arbre est identifié de manière unique par le couple (strategic_node, tree_seqnum).

4.4.1 Génération du message Tree Status

Un noeud racine d'un arbre ne génère jamais de Tree Status pour cet arbre.

Emission du Tree Status : Tout noeud n'envoie qu'un seul message Tree Status par numéro de séquence de message STC et numéro de séquence d'arbre. Néanmoins, pour assurer une tolérance aux pertes, dans le cas où le père d'un noeud a des flags différents de ceux du noeud, ce noeud répète son message. Si le noeud reçoit un message STC de son père avec le champ *flags* différent de celui qu'il maintient localement, alors le noeud répète son message Tree Status.
Plus généralement, chaque fois que le champ *flags* change, le noeud envoie un message Tree Status. Les flags envoyés dans le message Tree Status contiennent :

- FLAG_INCONSISTENT_STABILITY : vaut 1 s'il y a une incohérence au niveau de la stabilité (noeud non stable alors que la racine est stable). Ceci permettra à la racine d'éviter de démarrer le processus de coloriage de l'arbre alors que les liens de cet arbre ne sont pas stables.
- FLAG_STABLE : vaut 1 si la condition de stabilité est valide. Lorsque la racine reçoit un message Tree Status de chacun de ses fils avec ce bit mis à 1, elle peut lancer le coloriage.
- FLAG_UNSTABLE : vaut 1 si la condition de stabilité n'est pas valide.
- FLAG_TREE_CHANGE : vaut 1 s'il y a un changement des liens dans l'arbre (changement du père ou d'un fils). Dans ce cas, l'accélération de l'envoi des messages STC est utile.
- FLAG_NEW_NEIGHBOR : vaut 1 si le noeud découvre un nouveau lien symétrique. Cette information est utile par ce qu'elle permet d'assurer une cohérence de voisinge. De plus, l'apparition d'un noeud qui n'était pas pris en compte dans l'algorithme de coloriage peut invalider le coloriage. La racine pourrait alors incrémenter le numéro de séquence de l'arbre et lancer un nouveau coloriage.

Notons que il y a des changements de topologie qui ne sont pas pris en considérantion au niveau de ces flags, telle que la disparition d'un voisin, et l'expiration d'un voisin en tant que fils. La raison est que ces changements n'invalident pas le coloriage. Néanmoins, la mise à jour des données relatives à ces noeuds disparus est prévue au niveau des 2 modules de EOLSR, et au niveau de SERENA également (supprimer le tuple d'un voisin disparu par exemple).

Pour générer un message Tree Status, il faut remplir chaque champ dans le message comme suit :

- Message Type = Tree Status
- Message Size = taille du message en octets
- Sender Address = thisnode_address
- Message Sequence Number = S_seqnum
- Strategic Node Address = S_strategic_addr racine de l'arbre
- Numéro de séquence de l'arbre serena_info.S_treeseqnum
- Mettre les flags serea_info.S_stable
- Descendant Number = nombre de descendants de ce noeud dans cet arbre. C'est la somme du nombre de descendants des fils stables de ce noeud +1.

4.4.2 Réception d'un message Tree Status

Un noeud racine d'un arbre ne propage jamais de Tree Status pour cet arbre. Sur réception d'un message Tree Status, un noeud procède comme suit :

1. Si l'expéditeur de ce message n'appartient pas à l'ensemble des voisins symétriques alors le message doit être supprimé, exit.
2. Sinon, s'il n'existe aucun tuple correspondant à l'arbre (strategic_node_addr, tree_seqnum) Alors le message doit être supprimé, exit.
3. Sinon, si le numéro de séquence de l'arbre Tree Seq Num reçu dans le message < S_tree_seqnum stocké alors ignorer le message, exit. /*message avec un ancien numéro de séquence de l'arbre*/
4. Sinon, si le numéro de séquence de l'arbre Tree Seq Num reçu dans le message > S_tree_seqnum stocké alors le noeud conclut que la racine de cet arbre a incrémenté le numéro de séquence de l'arbre et a lancé un nouveau coloriage. Par conséquent :
   - le noeud arrête le coloriage en cours,
   - met à jour le temps de stabilité :
     stability_time = current_time + TREE_STABILITY_TIME
5. sinon,
   - mettre à jour les flags de stabilité.
   - si Sender Address == S_parent_addr alors si le champ 'flags' reçu dans le message est égal aux flags du noeud alors considérer le message comme acquittement des flags envoyés et donc le noeud fait : should_generate_tree_status = False, exit.
   - si Sender Address existe dans la table des fils (l'émetteur est un fils *child*) alors
     - child.c_descendant = Descendant Number envoyé dans le message
     - si (FLAG_STABLE == stable) alors :
       - child.status = stable
       - si la condition de stabilité locale est vérifiée alors mettre le status du noeud à stable et si le noeud n'est pas racine de l'arbre alors should_generate_tree_status = True

– si les flags de stabilité ont changé, alors si le noeud n'est pas racine de l'arbre alors il faut émettre un message Tree Status : should_generate_tree_status = True.

5 Routage

L'envoi et la transmission du message Hello et STC permet aux noeuds du réseau d'obtenir les informations nécessaires pour le routage. Les messages Hellos permettent aux noeuds de construire la table de voisins, et les messages STC générés par un noeud stratégique permettent aux noeuds de connaitre leur parent dans l'arbre dont la racine est ce noeud stratégique. Ce père représente le saut prochain pour atteindre la racine de l'arbre.

A la réception d'un paquet de données, le noeud routeur lit l'adresse de destination et détermine l'adresse du prochain saut next_hop lui permettant d'atteindre cette destination :
- Si l'adresse de destination est égale à son adresse alors next_hop = thisnode_addr,
- sion, si l'adresse destination est un voisin symétrique, next_hop = ce voisin,
- sinon, si l'adresse destination finale du paquet correspond à un noeud stratégique qui existe dans sa table STT, alors next_hop = S_parent_addr relatif à ce noeud stratégique.

Tout message généré par un noeud feuille est transmis au coordinateur auquel cette feuille est attachée. Il suffit donc que chaque feuille maintienne l'adresse de son coordinateur.

6 Changements de voisinage ou de parent/enfant dans un arbre

Les changements pouvant survenir dans la table de voisinage ou dans la table STT nécessitent :
- un traitement général indépendant du fait que l'arbre doit être colorié ou non,
- un traitement additionnel effectué uniquement si l'arbre doit être colorié.

6.1 Apparition d'un nouveau voisin symétrique

L'apparition d'un nouveau voisin symétrique est détectée lors de la réception d'un Hello qui fait passer à SYM le lien avec ce nouveau noeud. Le traitement à effectuer est le suivant :
- Le traitement général consiste à inclure ce voisin dans la table de routage si celle-ci est maintenue.
- Le traitement additionnel en cas d'arbre devant être colorié est la mise à 1 des flag FLAG_NEW_NEIGH et FLAG_STABLE. Par conséquent, ce changement de flags provoque l'émission du message Tree Status.
  Ceci permet d'éviter, dans la cas où le coloriage n'a pas encore démarré, un coloriage invalide réalisé sans tenir compte de ce nouveau lien.

6.2 Disparition d'un voisin symétrique

La disparition d'un voisin symétrique peut survenir :
- soit sur réception d'un Hello avec puissance insuffisante,
- soit sur expiration du timer N_SYM_time.

Le traitement à effectuer est le suivant :

1. traitement général :
   - le voisin disparu est effacé de la table de voisinage ;
   - si ce voisin disparu est un parent dans un arbre ;
     - le voisin disparu est effacé de la table STT
     - si l'option backup est utilisée et le backup est toujours valide alors remplacer tous les champs du parent par ceux du backup et effacer les champs du backup

2. traitement additionnel réalisé si l'arbre doit être colorié :
   - le voisin disparu est effacé de la table STT comme fils s'il apparait dans un arbre comme fils
   - le module SERENA est notifiée également de ce changement de topologie, afin de supprimer ce voisin.
   - remarquons que la disparition d'un voisin n'invalide jamais un coloriage.

6.3 Expiration du tuple de la table STT

L'expiration d'un tuple dans la table STT, se manifeste par un S_validity_time expiré. Le traitement général à effectuer est le suivant :
- si l'option backup est utilisée et le backup est toujours valide :
  - remplacer tous les champs du parent par ceux du backup
  - effacer les champs du backup
- sinon effacer le tuple de la table STT

6.4 Expiration du tuple d'un enfant dans la table STT

L'expiration du tuple d'un enfant dans la table STT, se manifeste par un C_validity_time expiré. Le traitement à effectuer est le suivant :

1. traitement général : rien car ce cas ne se pose pas puisque les tuples des enfants ne sont maintenus que pour les arbres devant être coloriés.
2. traitement additionnel : effacer le tuple associé à cet enfant.

7 Paramètres d'EOLSR

Dans EOLSR, les paramètres suivants sont définis

- Paramètres relatifs à l'état énergétique des noeuds :
  - MIN_ENERGY_LOW : une valeur minimale d'énergie résiduelle en dessus de laquelle un noeud de la classe Medium appartient désormais à la classe Low.
  - MIN_ENERGY_HIGH : une valeur minimale d'énergie résiduelle pour qu'un noeud puisse initialement appartenir à la classe Medium.

- Paramètres relatifs à la qualité des liens :
  - HYST_THRESHOLD_LOW : seuil minimum pour conserver un lien déjà accepté.
  - HYST_THRESHOLD_HIGH : seuil minimum pour accepter un nouveau lien.

- Paramètres relatifs à la périodicité des Hellos :
  - HelloMin : périodicité minimum des Hellos.
  - HelloMax : périodicité maximum des Hellos.

- Paramètres relatifs à la périodicité des STC :
  - STCMin : périodicité minimum des STCs.
  - STCMax : périodicité maximum des STCs.

- Paramètres relatifs à la validité des informations
  - NEIGHB_HOLD_TIME : le temps de validité des informations dans la table voisinage.
  - TREE_HOLD_TIME : le temps de validité des informations dans la table des arbres STT.

- STABILITY_TIME : le temps au bout duquel le voisinage est considére stable.

- Paramètres utilisés pour le calcul du coût d'une route
  - α_{High} : une constante =1
  - α_{Medium} : une constante = 4
  - α_{Low} : une constante =20
  - λ : une constante = 16 par exemple
  - β : une constante utilisée pour calculer le coût d'une route selon la formule donnée par les équations 3, 4 et 5. Cette constante est calculée comme suit :

$$\beta = \frac{\lambda * P_{receive}}{P_{trans}}. \tag{9}$$

  - MAX_HOP : nombre de sauts maximum qu'un message est autorisé à effectuer. Lorsque ce nombre est atteint, ce message est détruit.

8 Références

[1] C. Adjih, T. Clausen, P. Jacquet, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayum, L. Viennot, Optimized Link state Routing Protocol, RFC 3626, IETF, October 2003.

[2] L. Feeney, M. Nilson, Investigating the energy consumption of a wireless network interface in an ad hoc networking environment, IEEE INFOCOM 2001, Anchorage, Alaska, April 2001.

[3] S. Mahfoudh, P. Minet, Performance evaluation of the SERENA algorithm to SchEdule RoutEr Nodes Activity in wireless ad hoc and sensor networks, AINA 2008, IEEE 22nd International Conference on Avanced Information Networking and Applications, Ginowan, Japan, March 2008.

[4] S. Mahfoudh, P. Minet, An energy efficient routing based on OLSR in wireless ad hoc and sensor networks, PAEWN 2008, IEEE International Workshop on Performance Analysis and Enhancement of Wireless Networks, Ginowan, Japan, March 2008.